The exam will be in class, this Thursday, Apr 13. It will be closed-book, closed-notes, but you will be allowed two “cheat sheets” of notes, front and back.

So far, we have studied a wide variety of data structures for a diverse set of applications. We have considered the material from both a theoretical and practical perspectives. We have illustrated various aspects of data-structure design and analysis, including worst-case and asymptotic analyses, randomized data structures.

From the First Midterm: The exam is cumulative in theory, but the emphasis will be on material from after the midterm. Nonetheless, some questions (perhaps 20%) may be drawn from before the first midterm. Here is a summary:

**Basic Data Structures:** Sequential and linked allocation, amortized analysis, multilists and sparse matrices.

**Amortized Analysis:** Rather than analyzing the running time of each individual operation, we instead consider the average running time for a sequence of operations.

**Trees:** Representations of rooted trees, binary trees and traversals, full and extended binary trees, threaded binary trees, complete binary trees (and array allocation).

**Disjoint Set Union/Find:** A tree-based data structure for maintaining a collection of disjoint sets, and supports the operations union and find. Very fast amortized running time.

**Priority Queues:** We studied the binary heap and the leftist heap.

**Ordered Dictionaries:** Support the operations of insert, delete, and find, and various ordered extensions of these operations (e.g., find-up, get-min, range queries).

**Binary Search Trees:** Standard (unbalanced) binary search trees. Good expected-case performance ($O(\log n)$) for random insertions.

**AVL Trees:** Height-balanced trees. All dictionary operations in $O(\log n)$ time (worst case).

**2-3 Trees:** Variable-width nodes. Also $O(\log n)$ worst-case time.

**Red-Black and AA Trees:** Binary encodings of 2-3-4 and 2-3 trees. Also $O(\log n)$ worst-case time.

**Quad- and kd-Trees:** Partition trees for geometric point data based on axis parallel cuts. We studied operations on kd-trees in detail.

**Insertion:** (Unbalanced) insertion leads to $O(\log n)$ height in expectation if insertion order is random.

**Deletion:** (Unbalanced) Similar to binary-tree deletion, but using a complex process for finding replacement nodes.

**Orthogonal range queries:** Counting queries can be answered in $O(\sqrt{n})$ time in $\mathbb{R}^2$ and generally $O(n^{1-1/d})$ in dimension $d$. Reporting queries can be answered as well, adding an additional term to account for the number of points reported (e.g., $O(\sqrt{n} + k)$).
**Scapegoat Trees:** A tree that uses subtree-rebuilding to rebalance subtrees. It achieves $O(\log n)$ time for finds (worst case) and $O(\log n)$ time for insertions and deletions (amortized).

**Splay Trees:** A self-adjusting data structure. Based on an operation, called *splay*, which performs a series of rotations to bring a node to the root of the tree. Operations are based on splaying relevant nodes to (or near) the root and then performing operations near the root. Good amortized performance for many operations, particularly those that involve non-uniform access probabilities.

**Skip lists:** A multi-layered randomized variant of linked lists. Achieves $O(\log n)$ expected-case time for all dictionary operations (over random choices). Excellent performance both in theory and in practice.